TITLE:

トーリックイデアルに対する F_4,F_5アルゴリズムの解析

AUTHOR(S):

中山, 裕貴

CITATION:

中山, 裕貴. トーリックイデアルに対するF_4,F_5アルゴリズムの解析. 数理解析研究所講究録 2004, 1395: 16-23

ISSUE DATE:

2004-10

URL:

http://hdl.handle.net/2433/25921

RIGHT:

京都大学学術情報リポジトリ KURENAI 紅
Kyoto University Research Information Repository

数理解析研究所講究録 1395 巻 2004 年 16-23

16

トーリックイデアルに対するF_4,F_5アルゴリズムの解析

中山 裕貴
東京大学大学院情報理工学系研究科コンピュータ科学専攻*
NAKAYAMA HIROKI
DEPARTMENT OF COMPUTER SCIENCE, GRADUATE SCHOOL OF INFORMATION SCIENCE AND TECHNOLOGY, UNIVERSITY OF TOKYO

Abstract

近年，従来のブッフバーガーの算法とは異なるアプローチによる，F_4,F_5 アルゴリズムが Faugère により提案されている．本研究では，F_4,F_5 をトーリックイデアルに適用する．F_4 についてはトーリックイデアルに特化した改良を行い，また F_5 は Asir 上で新規に実装し，従来のブッフバーガーのアルゴリズムと比較する計算機実験を行った．その結果，項順序が全次数逆辞書式順序のときは F_4 および F_5 アルゴリズムは従来のアルゴリズムと比べ高速化されたが，項順序が辞書式順序の時は逆に遅くなってしまうことが観察された．この原因について，考察を行う．

1 背景

トーリックイデアルのグレブナ基底は，その離散性により，整数計画問題 [3] や組合わせ論など，様々な分野に応用されている．一方グレブナ基底を求めるアルゴリズムとしては，従来はブッフバーガーによる手法 [2] およびその改良版 [6]′が用いられていたが，最近になって Faugère により F_4[4] および F_5[5] アルゴリズムが発表され，注目を集めている．F_4 アルゴリズムは，複数の多項式を行列の形で簡約することで高速化を図っているが，イデアルがトーリックな場合，行列は非常に疎となり，逆に遅くなってしまう．一方の F_5 アルゴリズムは，新たな多項式の生成に用いた多項式の情報を保持することにより，不要な多項式の生成を完全になくすものであるが，今までに十分な計算機実験はなされていない．

今回は，この 2 つのアルゴリズムをトーリックイデアルに対して適用し，解析を行う．これにより，グレブナ基底計算の高速化の他に，トーリックイデアルという単純なケースについて，F_4,F_5 アルゴリズムの解析をより深く行うことができると期待される．特に，F_4 については，トーリックイデアルの性質を活かし，行列を有向グラフとみなすことで，実行時間・メモリの改善を達成することができる．

2 予備知識

多項式環 $k[\mathbf{x}]=k[x_1,\ldots,x_n]$($k$ は体，n は変数の数) において，指数ベクトル $\mathbf{a}=(a_1,\ldots,a_n)\in\mathbb{N}^n$ に対し $\mathbf{x}^{\mathbf{a}}=x_1^{a_1}x_2^{a_2}\cdots x_n^{a_n}$ と表す．係数が整数である行列 $A\in\mathbb{Z}^{d\times n}$ に対し，そのトーリックイデアル I_A は以下で生成される多項式イデアルのことである．

$$I_A=\langle \mathbf{x}^{\mathbf{u}}-\mathbf{x}^{\mathbf{v}} \mid A\mathbf{u}=A\mathbf{v} \quad \mathbf{u},\mathbf{v}\in\mathbb{N}^n\rangle$$

具体的に行列 $A\in\mathbb{Z}^{d\times n}$ が与えられたとき，イデアルの生成元は Robbiano らのアルゴリズム [1] によって求めることができる．

*nak-den@is.s.u-tokyo.ac.jp

続いて，項順序 $\succ$ を定める．多項式 $f \in I_A$ について，f の項で $\succ$ について最大となる項を f の主項 (initial term) といい，$in_\succ(f)$ と書く．トーリックイデアルのグレブナ基底 $G_\succ(I_A)$ は，以下の性質を満たす．

定義 1
有限集合 $G_\succ(I_A) = \{g_1, \ldots, g_s\} \subseteq I_A$ が $in_\succ(I_A) = \langle in_\succ(g_1), \ldots, in_\succ(g_s)\rangle$ を満たすとき，$G_\succ(I_A)$ を I_A の $\succ$ に対するグレブナ基底という．特に，任意の i に対し $in_\succ(g_i)$ の係数が 1 であり，かつ g_i のどの項も $in_\succ(g_j)\ (i \neq j)$ で割れないとき，$G_\succ(I_A)$ は被約 (reduced) グレブナ基底であるという．

グレブナ基底を求める方法として，以下のブッフバーガーのアルゴリズムが存在する．

アルゴリズム 1 (ブッフバーガーのアルゴリズム)
入力: イデアルの生成元 $F = \{f_1, \ldots, f_s\} \subset k[x_1, \ldots, x_n]$ および項順序 $\succ$
出力: グレブナ基底 $G = G_\succ(I_A)$
$G = F;$
do {
  $G' = G;$
  for each pair $\{p, q\}, p \neq q \in G'$ {
    $S = \overline{Spol(p,q)}^{G'};$
    if $S \neq 0$ *then* $G = G \cup \{S\};$
  }
} *while* $(G \neq G')$

このアルゴリズムは単純であるが，欠点として

- 新しい多項式が 1 つ生成されたとき，ペアの候補は今までに生成された多項式の数だけ増加するため，アルゴリズム全体ではペア (p, q) の数が指数的に増大する．
- 生成された多項式を簡約するとき，簡約に用いる多項式の候補として今までに生成された全ての多項式について除算を試すため，時間がかかる．

ことがあり，次に述べる F_4 および F_5 アルゴリズムによる改善が提案されている．

3 F_4 アルゴリズム

F_4 アルゴリズムでは，あらかじめ複数の S-多項式を生成し（例えば，sugar の値が最小になるものを全て選ぶ），それらを行列の形で同時に簡約を行う．これにより，多項式の簡約が高速化される．

3.1 Symbolic Preprocessing

まず，S-多項式の集合を F，今までに生成された多項式の集合を G とおくと，F の要素を簡約する可能性のある G の要素 (の単項式倍) Red は，以下のアルゴリズム (Symbolic Preprocessing) で求められる．

アルゴリズム 2 (Symbolic Preprocessing)
入力: S-多項式の集合 F, 多項式の集合 G
出力: $Red = \{ah \mid a$ は単項式 $, h \in G\}$
$T = \bigcup_{f \in F} T(f);$
$Red = \emptyset;$

while $(\exists g \in G, \exists t \in T \mid in_{\succ}(g)$ divides $t)$
  $Red = Red \cup \{t/in_{\succ}(g) \cdot g\}$;
  $T = (T \setminus \{t\}) \cup T(reductum(t/in_{\succ}(g) \cdot g))$;
}

ここで，$T(f)$ は f 中に出現する全ての項，$reductum(f)$ は f から主項を除いた多項式を表す．

このアルゴリズムにより，F を簡約する可能性のある多項式の集合を前もって（実際に簡約を行わずに）得ることができる．また，このとき同時に多項式を単項式倍しているため，次に述べる簡約ステップで，加減算のみで簡約を行うことができる．

3.2 行列による簡約

$F \cup Red$ に現れる全ての項の集合を T とし，T の元を項順序 $\succ$ の高い順に並べたものを $t_1 \succ t_2 \succ \cdots$ とする．多項式 f が $\sum_i a_i t_i : a_i \in k$ (k は体) と表されるとき，f に行ベクトル $(a_1\ a_2\ \cdots)$ を対応させる．逆に，行ベクトル v に対する多項式を $poly(v)$ と書く．

$$\begin{aligned} f_i &= (f_{i1}\ f_{i2}\ \cdots) \quad poly(f_i) \in F,\ f_{ik} \in k \\ r_j &= (r_{j1}\ r_{j2}\ \cdots) \quad poly(r_j) \in Red,\ r_{jk} \in k \end{aligned}$$

とするとき，左下の行列 A として

$$A = \begin{pmatrix} f_{11} & f_{12} & \cdots \\ f_{21} & f_{22} & \cdots \\ & \cdots & \\ r_{11} & r_{12} & \cdots \\ r_{21} & r_{22} & \cdots \\ & \cdots & \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 0 & ? & ? & ? & 0 & \cdots \\ 0 & 1 & ? & ? & ? & 0 & \cdots \\ 0 & 0 & 0 & \cdots & 0 & 1 & \cdots \\ & & & \cdots & & & \\ 0 & 0 & 0 & \cdots & 0 & 0 & \cdots \end{pmatrix}$$

とおく．これを，行に関する基本変形により，右上のような (対角) 行列 B に変形する．この行列 B は，以下の性質を持つ．

- $poly(B_i) \neq 0$ のとき，$poly(B_k)(k \neq i)$ は $in_{\succ}(poly(B_i))$ を含まない．

このとき，$poly(B_i)$ のうち，$in_{\succ}(poly(B_i))$ が $\{in_{\succ}(r) \mid r \in Red\}$ に含まれないものの集合が，F の各要素を G で簡約した正規形の集合となる．

3.3 F_4 アルゴリズムの改善

F_4 アルゴリズムにより，従来のブッフバーガーのアルゴリズムよりも 10 倍程度高速化された例が確認されている [7]．一方で，対象がトーリックイデアルの場合は，以下の原因により，逆に効率が悪くなる．

- 行列の各行には，1 が 1 つ，-1 が 1 つだけ含まれ，他は全て 0 である．
- つまり，行列は非常に疎であり，メモリを浪費する．
- 行列の基本変形には時間がかかる．

このため，トーリックイデアルに特化し，以下のようにデータ構造の改善を行う．

- 行列 $A \in \mathbb{Z}^{d \times n}$ に対して，n 点からなる無向グラフを考える．グラフの各点は番号を持つ．
- A の各行について，その第 i 列目が 1，第 j 列目が -1 のとき，点 (i,j) 間を辺で結ぶ．

そして，基本変形アルゴリズムを以下のように変更する．

1. 行列 A の各行から，d 個の辺からなる無向グラフを構成する．
2. 1. で得られたグラフに対し，連結成分分解を行う．
3. 2. で得られた各連結成分 $G_1, \ldots, G_i$ に対し，G_i の中で最も番号の大きい点を g_i とする．各連結成分 G_i に対し，G_i に含まれる g_i 以外の全ての点と g_i を辺で結ぶ．

このとき，得られたグラフが対角化された行列に対応している．このアルゴリズムは，枝数 (= 行列 A の行の数) の線形時間で行うことができ，行列の対角化よりも効率が良い．

例 1 行列 $A = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 \\ 1 & 0 & 0 & 0 & -1 & 0 \end{pmatrix}$ に対して，下図 1 の左側のグラフが構成される．このグラフの連結成分は $\{\{1,2,5\},\{3,4,6\}\}$ である．

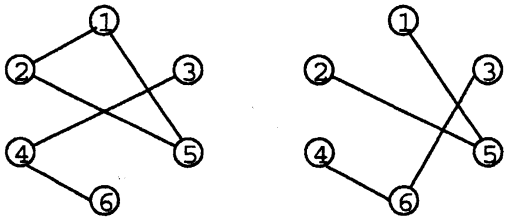

図 1: 行列 A のグラフ (左) とそれを変形した結果 (右)

左のグラフを上記の基本変形アルゴリズムで変形することで，上図右のグラフを得る．これが所望の対角行列を表しており，実際対応する行列

$$A' = \begin{pmatrix} 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

は，A を対角化したものとなっている．

4 F_5 アルゴリズム

F_5 アルゴリズムでは，多項式を新たに生成するとき，「どの多項式を用いて生成されたか」という情報 (signature) を多項式に付加する．この情報を用いることで，無駄な (0 に簡約される) 多項式の生成を完全に除去することができる．この節では，後に述べる $k[x_1, \ldots, x_n]^m$ 中の順序と区別するため，多項式 f の主項 $in_<(f)$ を $\mathrm{HT}(f)$ で表す．

定義 2

$f_1,\ldots,f_m$ を $k[x_1,\ldots,x_n]$ 上の多項式とする．$\mathbf{F}_i$ を $k[x_1,\ldots,x_n]^m$ 中の i 番目の単位ベクトルとし，関数 v を以下のように定義する．

$$v\left(\begin{array}{ccc} k[x_1,\ldots,x_n]^m & \longrightarrow & k[x_1,\ldots,x_n] \\ \mathbf{g}=(g_1,\ldots,g_m) & \mapsto & \sum_{i=1}^m f_i g_i \end{array}\right)$$

これより，$v(\mathbf{F}_i)=f_i$,$\mathbf{g}=\sum_{i=1}^m g_i\mathbf{F}_i$ となる．また，$k[x_1,\ldots,x_n]^m$ の間に順序 $\prec$ を以下の通り定義する．

$$\sum_{k=i}^m g_k\mathbf{F}_k \prec \sum_{k=j}^m h_k\mathbf{F}_k \text{ iff } (i>j \text{ and } h_j\neq 0) \text{ or } (i=j \text{ and } \mathrm{HT}(g_i)<\mathrm{HT}(h_i))$$

$\mathbf{g}$ の index i を「$\mathbf{g}=\sum_{k=i}^m g_k\mathbf{F}_k$ かつ $g_i\neq 0$ を満たす i」と定めると，順序 $\prec$ に対し，$in_\prec(\mathbf{g})=\mathrm{HT}(g_i)\mathbf{F}_i$ となる．また，$\langle f_1,\ldots,f_m\rangle$ に含まれる多項式の主項 t に対し，$W(t)=\{\mathbf{g}\in k[x_1,\ldots,x_n]^m \mid \mathrm{HT}(v(\mathbf{g}))=t\}$ とおく．

定義 3

関数 ω を $(t\mapsto \min_\prec W(t))$ で定義する．このとき，多項式 p に対し，p の **signature** $\mathcal{S}(r)$ を $in_\prec(\omega(\mathrm{HT}(p)))$ で定義する．

F_5 アルゴリズムでは，多項式 f_i の代わりに，signature と多項式のペア $r_i=(\mathcal{S}(r_i),poly(r_i)=f_i)$ を用いる．このようなペアの集合を R と書く．

定義 4

P を R の有限集合とし，$r\in R, t\in R$ とする．$poly(r)$ を

$$f=poly(r)=\sum_{p\in P} m_p p \quad m_p\in k[x_1,\ldots,x_n]$$

と表すと，すべての $p\in P$ に対し $\mathrm{HT}(poly(t))>\mathrm{HT}(m_p\cdot poly(p))$ かつ $\mathcal{S}(t)\succeq m_p\cdot\mathcal{S}(p)$ が成り立つとき，これを r の *t-representation* といい，$f=o_P(t)$ と書く．

定義 5

$(r_i,r_j)\in R^2$ が **normalized** であることを，以下で帰納的に定義する．

- $r\in R$ の *signature* が $e\mathbf{F}_k$ であるとする．e が $\langle f_{k+1},\ldots,f_m\rangle$ でこれ以上簡約されないとき，r は *normalized* であるという[1)]．
- $ur=(u\mathcal{S}(r),u\cdot poly(r))$ が *normalized* であるとき，単項と R のペア (u,r) は *normalized* であるという．
- $r_i,r_j\in R$ とする．$\tau_{i,j}$ を $poly(r_i),poly(r_j)$ の主項同士の LCM，$u_i=\tau_{i,j}/\mathrm{HT}(poly(r_i))$，$u_j=\tau_{i,j}/\mathrm{HT}(poly(r_j))$ とする．このもとで (u_i,r_i) と (u_j,r_j) がともに normalized であるとき，$(r_i,r_j)\in R^2$ は *normalized* であるという．

以上をもとに，F_5 アルゴリズムで用いる criterion を構成する．

定理 6 (new criterion)

$F=\{f_1,\ldots,f_m\}$ を多項式のリストとする．$G=\{r_1,\ldots,r_N\}\in R^N$ が以下の 2 つの条件

[1)] F_5 では $\langle f_{k+1},\ldots,f_m\rangle$ のグレブナ基底が予め求まっているので，単に主項の除算チェックで済む

1. $F \subset poly(G)$ である.
2. (r_i, r_j) が *normalized* である全ての $(i,j) \in \{1,\ldots,N\}$ に対し, $Spol(g_i, g_j) = o_{G_1}(u_i r_i)$ を満たす. ただし, $G_1 = \{g_1,\ldots,g_N \mid g_i = poly(r_i)\}$, $u_i = \mathrm{LCM}(\mathrm{HT}(g_i), \mathrm{HT}(g_j))/\mathrm{HT}(g_i)$ である.

を満たすとき, G_1 は $\langle f_1,\ldots,f_m\rangle$ のグレブナ基底である.

定理の証明, およびこの criterion を実装した具体的なアルゴリズムについては, [5] を参照されたい.

5 計算機実験による結果

以下の行列 A_n について, あらかじめ CoCoA [1] を利用してトーリックイデアルの生成元を求め, それを Asir への入力として用いた.

- $A_n = \begin{pmatrix} 1 & 2 & \cdots & n \end{pmatrix}$ 生成元は, $\langle x_1^2 - x_2, x_1x_2 - x_3, \ldots, x_1x_{n-1} - x_n\rangle$

項順序としては, 全次数逆辞書式順序および辞書式順序を用いた. 問題のサイズ (行列の列数) を 10 から 40 まで変化させ, Asir の組み込み関数 (gr), 組み込みの F_4(dp_f4_main), 今回改善した F_4, Asir 上で新規に実装した F_5 の実行時間を計測した結果が以下のものである. gr,F_4 の括弧の中の時間は, 多項式の簡約にかかる時間 (F_4 の場合は, Symbolic Preprocessing にかかる時間+基本変形にかかる時間) を表す.

表 1: 全次数逆辞書式順序での A_n のトーリックイデアルのグレブナ基底の計算時間 (秒)

n のサイズ	組み込み関数 (gr)	改善前の F_4	改善後の F_4	F_5
10	0.04826(0.004)	0.3582(0.248)	0.04848(0.004)	0.1001
15	0.4298(0.044)	3.004(2.569)	0.255(0.032)	0.3944
20	1.935(0.266)	19.08(18.52)	1.044(0.171)	1.719
25	7.006(0.960)	74.86(71.16)	3.332(0.565)	5.685
30	20.97(2.438)	199.8(191.3)	9.276(1.650)	15.55
35	61.6(6.402)	764.8(750.3)	19.55(3.791)	37.48
40	155.6(14.07)	1832(1800)	42.14(8.086)	84.46

表 2: 辞書式順序での A_n のトーリックイデアルのグレブナ基底の計算時間 (秒)

n のサイズ	組み込み関数 (gr)	改善前の F_4	改善後の F_4	F_5
10	0.1342(0.010)	0.8391(0.672)	0.1304(0.020)	0.5125
15	0.709(0.193)	8.749(8.156)	0.6792(0.243)	12.92
20	2.771(0.645)	68.82(63.76)	3.292(1.490)	133.9
25	9.387(2.369)	322.2(315.8)	13.38(6.158)	803.8
30	26.23(6.301)	1336(1318)	40.65(18.84)	3341
35	70.28(16.23)	4578(4526)	113.4(53.70)	12350
40	168.7(36.04)	24320(24150)	273(127.9)	[too large]

続いて, F_4 アルゴリズムにおける 2 つの項順序の違いによる中間基底の経過の差を調べた. $n = 20$ の場合について, 簡約する S-多項式・簡約に用いる多項式・0 以外に簡約された S-多項式の数を以下の表にま

とめた．ここで，2つの項順序間のsugarの最大値の違いは，グレブナ基底の次数の違いに起因する（A_{20}のトーリックイデアルのグレブナ基底の最大次数は，全次数逆辞書式順序の場合は3，辞書式順序では20)．

表3: F_4による全次数逆辞書式順序での計算の途中経過

sugar	簡約するS-多項式の数	簡約に用いる多項式の数	0以外に簡約されたS-多項式の数
3	171	17	171
4	2109	373	0(終了)

表4: F_4による辞書式順序での計算の途中経過

sugar	簡約するS-多項式の数	簡約に用いる多項式の数	0以外に簡約されたS-多項式の数
3	171	17	171
4	2118	492	9
5	168	323	6
6～21	20～100程度	300～400程度	1～5
22	18	434	0(終了)

6 結論

本研究では，F_4のトーリックイデアルに特化した改善，およびF_5の実装を行った．まず，F_4を改善した結果，全体として数十倍程度の高速化が達成され，1行n列の行列A_nについては，項順序が全次数逆辞書式順序のとき，組み込み関数grよりも数倍高速になった．一方で，項順序が辞書式順序のときは，問題のサイズが大きくなるにつれ，組み込み関数grより遅くなる傾向が見られた．原因としては，全次数逆辞書式順序の場合はS-多項式の数に比べて簡約に用いる多項式の数の方がずっと少なく，「多数の多項式を少数の多項式で一度に簡約する」というF_4の利点が活かされているものの，逆に辞書式順序の場合は次数の高い多項式を生成するときに，ごく少数の多項式を，必要以上に多数の多項式で簡約しようとしているためである．

また，F_5に関しては，全次数逆辞書式順序のときは組み込み関数を上回ったものの，辞書式順序の時は非常に遅くなり，メモリ使用量も大きくなった．その理由として，ペアの選択戦略としてsugarを用いていないため，次数が上がるにつれ効率が非常に悪化することなどがあげられる。

参 考 文 献

[1] A.M. Bigatti, R. Scala, and L. Robbiano. Computing toric ideals. *Journal of Symbolic Computation*, 27:351–365, 1999.

[2] B. Buchberger, *Ein algorithmisches Kriterium für die Lösbarkeit eines algebraischen Geichungssystems*. Aequ. Math. 4/3(1970), 374–383.

[3] P. Conti and C. Traverso. Buchberger algorithm and integer programming. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes (AAECC-9, New Orleans, 1991)*, volumne 539 of *Lecture Notes in Computer Science*, pages 130–139, Berlin, 1991. Springer.

[4] J.C. Faugère. A new efficient algorithm for computing Gröbner bases (F_4). *Journal of Pure and Applied Algebra*, 139(1–3):61–88, 1999.

[5] J.C. Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero F_5. In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, pages 75–83, 2002.

[6] R. Gebauer and M. Möller. On all installation of Buchberger's algorithm. *Journal of Symbolic Computation* , 6/2/3:275–286, 1989.

[7] 野呂正行, 計算機代数入門, 2001